CS 271 - Project 0101

James Le - Kevin Ly

March 29, 2017

1. Prove the following by induction.

   (a) A complete binary tree with height h contains $2^{h+1} - 1$ total nodes.

   Solution: Proof by induction

   Base Case: $h = 0$. A binary tree of height 0 has one node. $2^{h+1} - 1$ equals one for $h = 0$. Therefore true for $h = 0$.

   Inductive Hypothesis: Assume that the number of nodes in a binary tree of height h is $2^{h+1} - 1$, for $h = 1, 2, ..., k$.

   Now consider a tree T of height $k + 1$. The root of T has a left subtree and a right subtree each of which has height at most k. These can have at most $2^{k+1} - 1$ nodes each by the induction hypothesis. Adding the root node gives the number of nodes in a binary tree of height $k + 1$ to be

$$2(2^{k+1} - 1) + 1 = 2 * 2^{k+1} - 2 + 1 = 2^{(k+1)+1} - 1$$

   (b) A complete binary tree with n nodes has $(n - 1)/2$ internal nodes.

   Solution: Proof by induction

   Base Case: A binary tree with a single node ($n = 1$) has no internal nodes. $(n-1)/2$ equals 0 for $n = 1$. Therefore true for $n = 1$.

   Inductive Hypothesis: Assume that the number of internal nodes in a binary tree with n nodes is $(n - 1)/2$ for $n = 1, 2, ..., k$.

   Now consider a tree T with $k + 1$ nodes. The root of T has a left subtree and a right subtree each of which has at most $\frac{k}{2}$ nodes. These can have at most $\frac{\frac{k}{2}-1}{2} = \frac{k-2}{4}$ nodes each by the induction hypothesis. Adding the root node gives the number of internal nodes in a binary tree with $k + 1$ nodes to be

$$2 * (\frac{k-2}{4}) + 1 = \frac{k-2}{2} + 1 = \frac{k}{2} = \frac{(k+1) - 1}{2}$$

2. Consider a binary search tree T whose keys are distinct. Prove that if the right subtree of a node x in T is empty and x has a successor y, then y is the lowest ancestor of x whose left child is also an ancestor of x.

   The following procedure returns the successor of a node x in a binary search tree if it exists, and NIL if x has the largest key in the tree:

```
TREE-SUCCESSOR(x)

if x.right != NIL

return TREE-MINIMUM(x.right)

y = x.p

while y != NIL and x == y.right
```

```
5   x = y
6   y = y.p
7 return y
```

We break the code for TREE-SUCCESSOR into 2 cases:

- If the right subtree of node x is nonempty, then the successor of x is just the leftmost node in x's right subtree, which we find in line 2 by calling TREE-MINIMUM(x.right).
- On the other hand, if the right subtree of node x is empty and x has a successor y, then y is the lowest ancestor of x whose left child is also an ancestor of x. To find y, we simply go up the tree from x until we encounter a node that is the left child of its parent; lines 3-7 of TREE-SUCCESSOR handle this case.